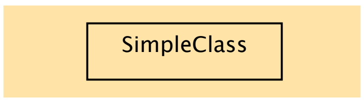
Basic UML Class Diagrams Cheat Sheet

In the simplest case, a class can simply be represented by a name inside a box. (By convention, class names always start with a capital letter.)



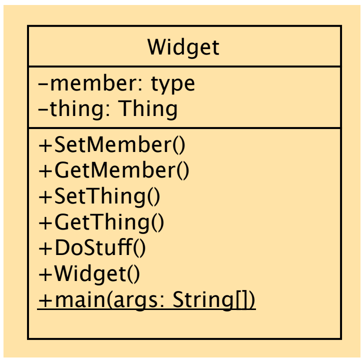
Typically, however, a class representation will include a list of member variables inside a pair of horizontal lines, followed by a list of methods.

The - and + symbols stand for “private” and “public” respectively.

If the type of a member variable starts with a capital letter, this refers to another class.

The Widget() method in this example is the class's constructor method

Underlined functions like main() are static and can be called without creating an instance of the class.



Relationships

Multiplicity

In relationships between instances, notation such as 1..* is used to indicate the number of instances on each side of the relationship. In the aggregation example below, a single MultiMessage instance (1) is associated with zero or more messages (0..*, or just *).

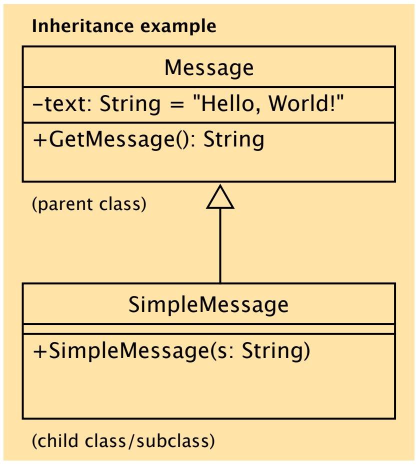
Generalization/Inheritance

Relationships are indicated by lines/arrows.

These can be broadly categorized into relationsips between the classes themselves (generalization/ inheritance) and relationships between the instances of these classes (association).

Inheritance (an “*is a*” relationship) is represented by a hollow arrow pointing from the child to the parent.

Multiplicity is not shown because this is a relationship between classes, not instances.

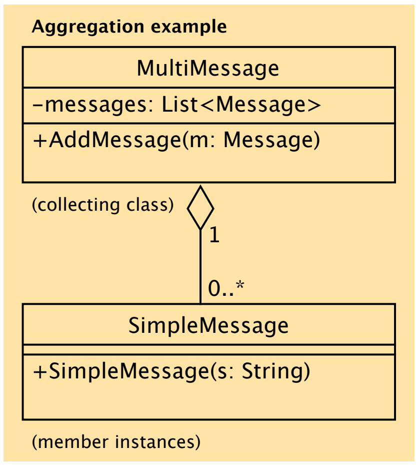


Aggregation

Aggregation is a weak kind of “*has a*” relationship where one class contains a collection of instances of other classes.

The key idea is that the existence of the member instances is not dependant on the existence of the collecting class instance.

The member instances must be created before they are added to the collection.

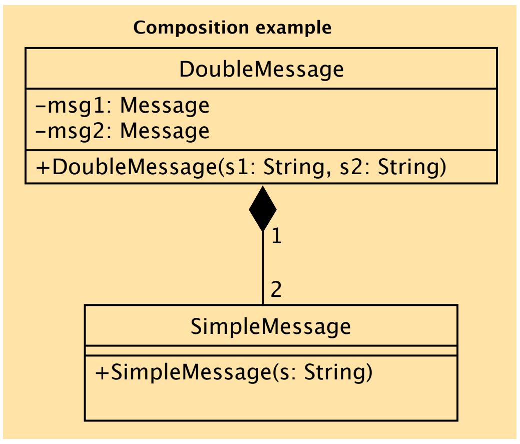


Composition

Composition is a stronger form of “*has a*” relationship where the member instances cannot exist without the composer class.

The member instances are typically created by the constructor function of the composer class, and destroyed by the destructor function of the composer class. There are no methods to manage the creation of component instances.

A composition relationship is represented by a filled diamond at the composing end.


Composition example
DoubleMessage
–msg1: Message
–msg2: Message
+DoubleMessage(s1: String, s2: String)
1
2
SimpleMessage
+SimpleMessage(s: String)